# Tour of common optimizations

# Simple example

```
foo(z) {
    x := 3 + 6;
    y := x - 5
    return z * y
}
```

# Simple example

```
Strength reduction
```

```
x := a + b;
y := a + b;
```

```
x := a + b;

only if x, a, b not

modified!

y := a + b; x
```

```
if (...) {
    a := read();
    x := a + b;
    print(x);
}
....
y := a + b;
```

```
if (...) {
    a := read();t:= α+b
    x := a+b;t
    print(x);
} the \{ t:= α+b \}

Y := a +b;t

    y := a +b;t
```

```
x := y
\vdots
z := z + x
```

```
x := y
z := z + xy
x,y \text{ and modified}
x := y
y + xy + y
y = y
y = y
```

```
x := y
z := z + y
```

What if we run CSE now?

$$x := y$$
 $z := z + \sqrt{x}$ 

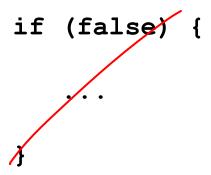
What if we run CSE now?

```
x := y**z
...
```

Often used as a clean-up pass

$$\mathbf{x} := \mathbf{y}$$
 $\mathbf{z} := \mathbf{z} + \mathbf{x}$ 
 $\mathbf{z} := \mathbf{z} + \mathbf{y}$ 
 $\mathbf{z} := \mathbf{z} + \mathbf{y}$ 
 $\mathbf{z} := \mathbf{z} + \mathbf{y}$ 
 $\mathbf{z} := \mathbf{z} + \mathbf{y}$ 

```
if (false) {
    ...
}
```



dead code clim (unreachable code clim)

Another comma clean up oft

• In Java:

```
a = new int [10];
for (index = 0; index < 10; index ++) {
   a[index] = 100;
}</pre>
```

In "lowered" Java:

```
a = new int [10];
for (index = 0; index < 10; index ++) {
   if (index < 0 || index >= a.length()) {
      throw OutOfBoundsException;
   }
   a[index] = 0;
}
```

In "lowered" Java:

```
a = new int [10];

for (index = 0; index < 10; index ++) {

   if (index < 0 || index >= a.length()) {
      throw OutOfBoundsException;

   index = 0;

   index = 0;
```

```
p := &x;
*p := 5
y := x + 1;
```

```
p := &x;

×*p := 5

y := x + 1; 6
```

pointe/alias analysis

```
x := 5;
*p := 3
y := x + 1; ???
```

```
for j := 1 to N
   for i := 1 to M
    a[i] := a[i] + b[j]
```

Loop invariant code motion

```
area(h,w) { return h * w }
h := ...;
w := 4;
a := area(h,w)
```

```
area (h, w) { return h * w }

h := ...;
w := 4;
a := area(h, w)
h := ...;
h
```

### Optimization themes

- Don't compute if you don't have to
  - unused assignment elimination
- Compute at compile-time if possible
  - constant folding, loop unrolling, inlining
- Compute it as few times as possible
  - CSE, PRE, PDE, loop invariant code motion
- Compute it as cheaply as possible
  - strength reduction
- Enable other optimizations
  - constant and copy prop, pointer analysis
- Compute it with as little code space as possible
  - unreachable code elimination